PA3 Balatro

CIIC3015

PROJECT DUE: Dec 8, 2025 11:59 PM

Partner Selection

Welcome to Project Assignment 3: Balatro!
Before continuing, we highly recommend that you choose a partner for this project as soon as possible. Choose wisely! For this project, you will need to form a group of **three people** (yourself and two partners). There is **no deadline for choosing your partners**. If you're having difficulty finding partners, you can either post a message in the Find a Partner channel on the class Discord server or contact the TA in charge of projects (anthony.torres25@upr.edu)

Introduction

Today, you'll be working on Balatro, the world famous roguelike poker game developed by LocalThunk. You've probably heard of it before but in case you haven't, Balatro is a deck building game where players create powerful poker hands to earn chips and defeat increasingly challenging blinds. After each round, the game evaluates your hand and provides feedback based on your score and multipliers. Hands determine your base score, multipliers amplify your score through joker and other effects. Chips represent your total points earned each round. The challenge of Balatro lies in combining strategy, luck and deck optimization to build powerful synergies and push for higher scores. This project is to finish implementing some mechanics of Balatro. Since this could be a complicated game to understand before continuing there are going to be a few other sections explaining the game in more detail.

Video explanation of how Balatro works:

- Balatro Explained In 5 Minutes: https://www.youtube.com/watch?v=EFEauLTZZJ0&t=165s

Understanding Balatro Pt. 1 - Jokers and Deck

One of Balatro's defining features is its Jokers, cards that provide unique effects or score multipliers. Players can equip multiple Jokers at once (up to 5) each drastically altering the way hands are played and scored. Some Jokers boost hands (example: high cards), others give chips per played, and some even change the rules of poker entirely. Since we wanted to make this a little bit more interesting the joker cards that you will find in the game are completely different that you will find in Balatro (card art). So here's a little table explaining what each Joker card does, you can always go back to this table when doing future tasks!

Joker	Effects	Price
The Joker	+4 multiplier	4
Micheal Myers	Adds a random multiplier between 0 and 23	6
Fibonacci	Each played Ace,2,3,5,8 gives +8 Mult	5
Gauntlet	+250 Chips, -2 hand size	5
Ogre	+3 Mult for each owned Joker (treat as number of jokers owned)	5
Straw Hat	+100 Chips then -5 chips for every hand already played this round	5
Hog Rider	+100 Chips if the played hand is a Straight	4
? Block	+4 Chips if the played hand used exactly 4 cards	5
Hogwarts	Each Ace played gives +4 mult and +20 chips	6
802	If this is the last hand (amountOfHands == 0 after decrement above), double the added amount	6

Planets	Effects	Price
Mercury	Increases High Card hand value by +1 multiplier and +10 chips	2
Venus	Increases One Pair hand value by +1 multiplier and +15 chips	2
Earth	Increases Two Pair hand value by +2 multiplier and +15 chips	2
Mars	Increases Three of a Kind hand value by +2 multiplier and +25 chips	2
Jupiter	Increases Straight hand value by +3 multiplier and +25 chips	3
Saturn	Increases Flush hand value by +3 multiplier and +30 chips	3
Uranus	Increases Full House hand value by +3 multiplier and +35 chips	3
Neptune	Increases Four of a Kind hand value by +4 multiplier and +40 chips	3
Sun	Increases all hands value by +2 multiplier and +30 chips	12

Understanding Balatro Pt. 2 - Progression System

Balatro is not just about making poker hands, it's also about surviving through a roguelike progression system.

🧠 **Understanding** *A roguelike is a game structured around failing and restarting, in which no two play-throughs are exactly alike due to a design that randomizes various aspects of the experience*

- ***NYTimes*** *(Don't Be Scared. Play a Roguelike.)*

Each Ante (level) has three rounds: the small Blind, the Big Blind, and a Boss Blind that introduces special challenges or restrictions. Players must score enough chips to beat each blind and advance to the next.

Boss Blind	Debuff
The Mark	All Face cards are drawn face down
The Needle	Play only 1 hand
The House	First hand is drawn face down
The Hook	Discards 2 random cards held in hand after every played hand
The Water	Start with 0 discards
The Manacle	-1 hand size

After each completed blind, the player will be rewarded with a certain amount of money, and the shop will appear. This is where the player can purchase new Jokers and Planet cards to enhance their deck!

Setup

Accept the Assignment

- Copy and paste the following link to accept / join PA3 https://classroom.github.com/a/oygXrkNd. One of the partners must create the group and the other partner must join it. DO NOT WORK ON THE SAME LAPTOP.

PyGame

- For this project you will need PyGame. To install it you can run pip install pygame in the Pycharm terminal or search the packages in pycharm as we did in PA0.

⚠️ **WARNING** *For all projects you will follow the same workflow as all previous projects. Meaning that you should commit and push all tasks individually.*

Let's Start Tackling Those To-Do's!

Part 0: The Prelude

Before you even think about tackling any of the tasks, you must create the game levels! Without the levels properly set up, the entire project will fail to run. Every time you attempt to start the game, it will throw an error, meaning no testing, no debugging, and definitely no progress. This isn't just a suggestion, it's mandatory! The levels form the foundation of the entire game's structure. They define the progression system, the blinds (Small, Big and Boss), and the unique challenges tied to each boss. Without them, the game has no roadmap to follow, leaving your code stranded before it even reaches the main loop. Every future task from calculating rewards, to buying Jokers, fighting Boss Blinds to giving cards to players depends on having a complete and functional level setup. A pretty cool helpful feature was added, I wonder how you could activate it?

Task 0: Creating the Levels 🛠️

Found in: LevelManager.py

It's to start creating those bosses. Set up all levels and their corresponding sublevels. Each level should include. Set up all levels and their corresponding sublevels. Each level should include a Small, Big, and Boss Blind, with Boss Blinds assigned unique names. Organize them in a dictionary called levelsDict where each key represents a level number.

💡 **Hint**

- Understand how the sublevel file works and how we could use it.
- The key of levelsDict represents the ante number or level number
- The value of levelsDict represents each sublevel for that ante (small blind, big blind and boss blind)

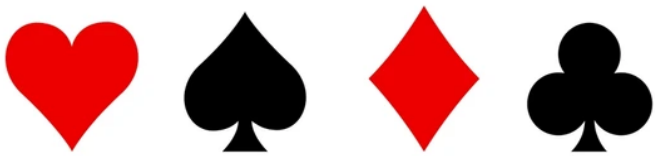
Part 1: The Cards

As you have probably already noticed, Balatro is a card game, so what's a card game with no cards to play with? That's what this part is about. You are going to be **implementing the fundamental logics for the cards**. From giving the cards to the player, to sorting the cards to indicate what type of hand the player played to discarding the cards. This part is fundamental and you have to do this task first before continuing on developing the other tasks of this project. Here I'm gonna give a little brief explanation of how the cards work and what the hands you have.

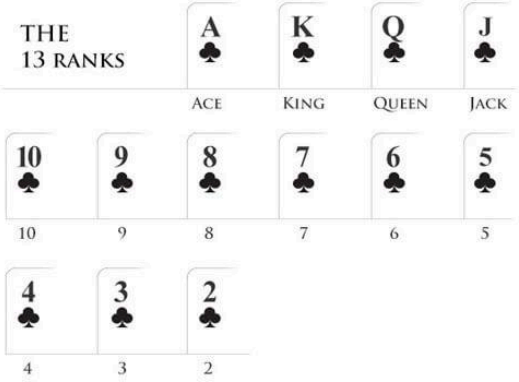
Understanding Suits and Ranks

In a standard deck of playing cards, ranks and suits are the two main attributes that define each card.

- **Suits**: There are four suits in a standard deck, each suit contains the same set of ranks:
    - Hearts
    - Diamonds
    - Clubs
    - Spades

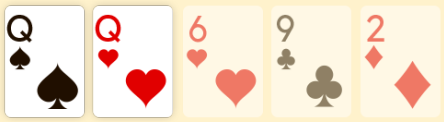


- **Ranks**: Each suit has 13 ranks, which represents the value of order of the cards
    - Number cards: 2-10
    - Face cards: Jack (J), Queen (Q), King (K)
    - Ace: Can be high or low depending on the game



As mentioned before, Balatro uses the poker basics like the playing cards the the hands you can play here you will find all of the hands you can play:

Hand	How to play it	Visual Images
Straight Flush	5 cards in a row (consecutive ranks) all sharing the same suit	
Four of a Kind	4 cards with the same rank	
Full House	3 cards sharing a rank an 2 other ones sharing another rank	
Flush	5 cards that share the same suit	
Straight	5 cards in a row (consecutive ranks)	
Three of a Kind	3 cards sharing a rank	
Two Pair	2 pairs of cards with different ranks	

One Pair	1 pair of cards with the same rank	
High Card	If the played hand is not on the list, only the highest rank scores	

You can find this information when running the project and going to run info. That's all the necessary information you need to know before continuing this part of the project!

Task 1: Giving the cards ♦️

Found in: DeckManager.py

One of the most important things in any playing card game is the ability to have cards to play them. So in this task you will have to **implement a function that creates a full deck of cards**. Loop through all possible suits and ranks, retrieve the corresponding image from the cardImages dictionary using (suit, rank) as the key, and create a card object for each valid combination. If a matching image is not found, skip that card. Add each created card to a list called `deck` and return the completed list at the end. You will be given a few variables to start off with cardImages and deck.

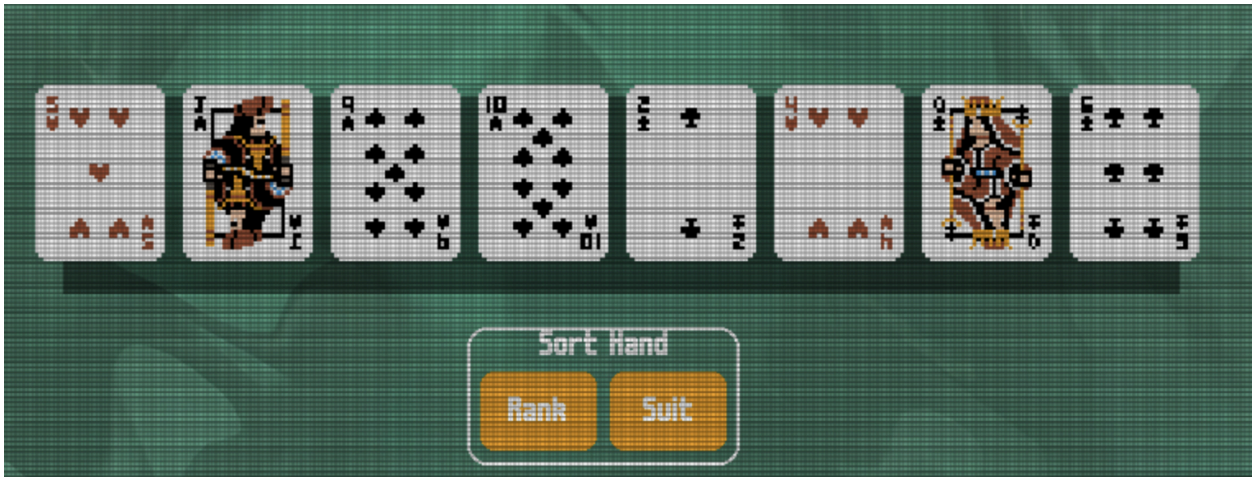
> 💡 **Important** *To add a card to the deck, do the following:*
> `deck.append(Card(suit=suit, rank=rank, image=image))`

Task 2: Sorting the cards ♣️

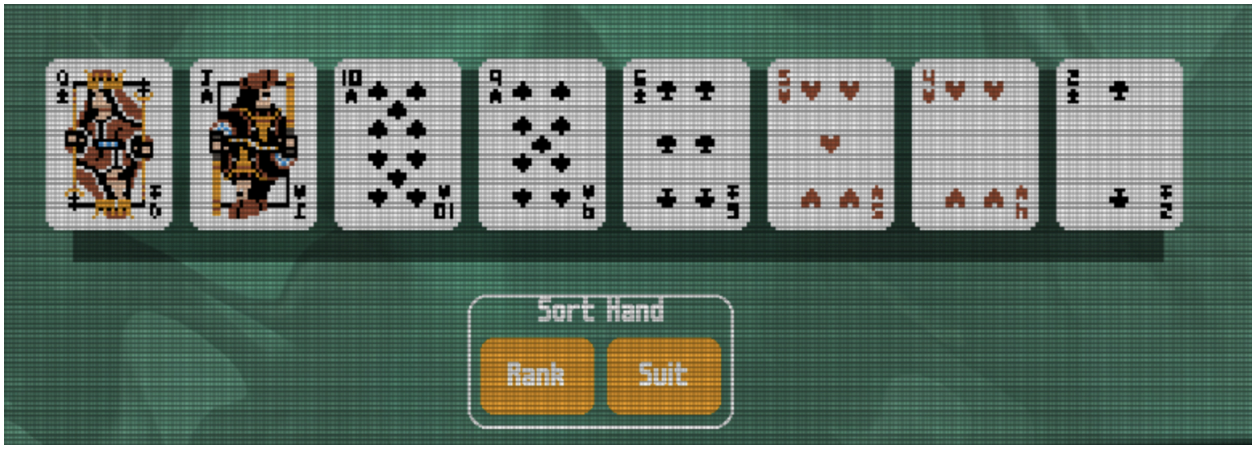
Found in: GameState.py

When you deal the cards to the player, the hand might look a bit disorganized. That's why there are two buttons below: one labeled "Sort by Rank" and another labeled "Sort by Suit."

Before Sorting:



After Sorting by rank:



After Sorting by suit:

Your task is to implement a basic card-sorting system **without using lambda functions or built-in sort methods**. Create a `suitOrder` list (Hearts, Clubs, Diamonds, Spades) and use **loops** to compare each card with the ones after it. Depending on the selected mode, sort the hand by rank first or suit first, swapping cards as needed until the entire hand is properly ordered.

Task 3: Playing the cards ♥

Found in: HandEvaluator.py

In any poker-style card game, it's essential to determine what kind of hand a player has such as a pair, flush, or straight. In this task, you will implement a function that evaluates a given hand of cards and returns the hand's rank as a string (for example, "Full House", "Flush", or "High Card"). You will be given a list of Card objects called hands. Each Card has two attributes: rank and suit. Your function should analyze these attributes to identify the best possible poker hand. Start by looping through all cards in hand and collecting their ranks and suits into separate lists. Use a dictionary to count how many times each rank appears (this will help detect pairs, three of a kind, or four of a kind). Sort these counts in descending order to easily identify the highest combinations. Next, use another dictionary to count how many times each suit appears. If any suit appears 5 or more times, mark the hand as a flush. To check for a straight, first remove duplicate ranks and sort the remaining ones. Then, look for sequences of five consecutive numbers. Use the rank counts and boolean flags to determine other possible hands in the following order: Four of a Kind, Full House, Flush, Straight, Three of a Kind, Two Pair, One Pair, High Card. Finally, return a string representing the best matching hand type. You will be given the variable hand as input, and your function should return the hand's evaluation as a string at the end.

👀 **Visualization** This is how the hand variable could look like:

hand = [

Card(Rank.ACE, "hearts"),

Card(Rank.KING, "hearts"),

Card(Rank.QUEEN, "hearts"),

Card(Rank.JACK, "hearts"),

Card(Rank.TEN, "hearts")

]

Task 4: Discarding the cards ♠

Found in: GameState.py

You have made pretty good progress so far! Let's say there is this card I don't want on my deck. What could I do about it right now? Nothing! This is where you come in.
You will have to repeatedly remove selected cards from the player's hand until no cards remain, then draw new cards to refill the hand. You must include a clear base case to stop the recursion when there are no more cards left to discard. You can use any helper functions but AVOID using for or while loops, all repetition must happen through **recursive** calls to `discardCards()` itself. Once all the cards have been discarded and new ones drawn, make sure the function resets the card selections, clears any related lists or text, and updates the hand display.

⚠️ **WARNING** *For full points **YOU MUST IMPLEMENT IT RECURSIVE***

💡 **Important** *To update the visuals* `self.updateCards(400, 520, self.cards, self.hand, scale=1.2)`

Part 2 The Wild Cards

Congratulations! You've successfully completed the core playing card mechanics! That's a big milestone, great job. I know some parts may have been a bit challenging, but now it's time to have some fun and bring more personality and strategy into the game by adding Jokers! In this next section, you'll be focusing on **implementing Jokers, assigning their prices, and defining their unique effects** that can change the flow of a run. Alongside that, you'll also be **introducing Planet Cards**, which come with their own special abilities that can influence gameplay in exciting ways. By the end of this part, you'll have a fully **functional shop system** where players can buy Jokers and Planet Cards to enhance their deck, adding depth, variety, and replayability to the overall experience.

❗ **Fun Fact** *Balatro is latin for a professional jester, buffoon, or clown*

Task 5: The Jokers 🃏

Task 5.1: Pricing the Jokers 💲

Found in: DeckManager.py

Jokers don't come for free! **Complete the `priceMap` variable by assigning each joker a price.** The key should represent the joker's name, and the value should be the joker's price. The price table can be found earlier in this document.

Task 5.2: Applying the Joker Effects 🔥

Found in: GameState.py

Let the Joker mayhem begin! It's time to fully bring out the Balatro side of Balatro. As you start implementing each Joker's unique effect, make sure to reference the Joker table. It contains the descriptions and mechanics that define how every Joker should influence the game. Each Joker must follow a consistent structure to keep the system clean and modular. Use the following format when adding any Joker's functionality:

💡 **Important**

```
if " joker card name " in owned:
    # the effects of that joker card
    self.activated_jokers.add("joker card name")
```

That final line is essential, it ensures the game correctly registers and visually displays which Jokers are currently active. This allows players to see which effects are influencing their run, maintaining clarity and feedback throughout gameplay. As you expand the roster of Jokers, always keep this format in mind so new additions blend seamlessly into the overall Balatro system.

Task 6: The Planets 🌎

Task 6.1: The System

Found in: Planets.py

Let's start implementing the Planet Card system for Balatro. You'll need to create a dictionary named `PLANETS` that stores all available planet cards as `PlanetCard` objects. Each entry in this dictionary should use the planet's name as the key and a corresponding PlanetCard instance as the value. Every planet must include its name, a short description of the poker hand it levels up, a purchase price, a chip bonus, and a multiplier value.

⚡ **Example** "Gusty Garden": PlanetCard("Gusty Garden", "levels up galaxy", 6, 15, 7)

Task 6.2: Applying the Planet Effects 🔥

Found in: ShopState.py

Now it's time to implement how Planet Cards affect the player's hand levels and scoring system. Loop through all entries in the `HAND_SCORES` dictionary and compare each hand type with the planet's effect. If the planet is not the Sun, check whether the hand name matches the one described in the planet's description (for example, "levels up Two Pair" → "Two Pair"). When a match is found, increase that hand's "chips", "multiplier", and "level" values based on the planet's stats, then break out of the loop to prevent unnecessary iterations. However, if the planet is the Sun, apply its bonuses universally by increasing "chips", "multiplier", and "level" for every hand type. This ensures regular planets upgrade one specific hand, while the Sun upgrades them all.

👀 **Visualization**

HANDS_SCORE variable holds:

```
HAND_SCORES = {
    "High Card": {"chips": 5, "multiplier": 1, "level": 1},
    "One Pair": {"chips": 10, "multiplier": 2, "level": 1},
    "Two Pair": {"chips": 20, "multiplier": 2, "level": 1},
    "Three of a Kind": {"chips": 30, "multiplier": 3, "level": 1},
    "Straight": {"chips": 40, "multiplier": 4, "level": 1},
    "Flush": {"chips": 50, "multiplier": 4, "level": 1},
    "Full House": {"chips": 60, "multiplier": 5, "level": 1},
    "Four of a Kind": {"chips": 80, "multiplier": 7, "level": 1},
    "Straight Flush": {"chips": 100, "multiplier": 8, "level": 1},
}
```

Part 3 Rougeliking it

The Balatro portion of the game has officially been integrated, great work! Now it's time to take things to the next level by adding the roguelike features that will bring depth and replayability to your project. From implementing the reward system that grants money to the player, to introducing Boss Blinds that challenge their strategy and adaptability, this final stage ties everything together. It's a true send-off task, one that marks the completion of your core gameplay loop and sets the foundation for endless runs and dynamic progression!

Task 7: Giving Money 💵

Found in: GameState.py

You're going to implement a function called `calculate_gold_reward()`, which determines how much gold a player earns after completing a blind. The base rewards are simple: 4 gold for a Small Blind, 8 for a Big Blind, and 10 for a Boss Blind. But here's where it gets interesting, if the player exceeds the target score, they'll earn bonus gold for their overkill performance. The higher they surpass the target, the more extra gold they'll receive. There's one catch: this entire function must be implemented using **recursion**, **no loops** allowed. That means your solution should repeatedly call itself to calculate both the base and bonus rewards, building up the final gold total step by step.

⚡ **Use this!** **Overkill Formula:**

$$\text{bonus} = \min(5, [\max(0, \frac{score - target}{target} \cdot 5])$$

Task 8: Checking Unfinished Levels 🕵️

Found in: LevelManager.py

When doing levels it's important to know our progress and how we are going to be progressing from that point. So to achieve this we need to be checking the unfinished levels. Create a **recursive** function that finds the next unfinished sublevel. It should check each sublevel in order and return the first one that isn't finished. Stop once all have been checked or one is found. **Avoid using loops and use recursions.**

Task 9: The Boss Blinds 👑

Task 9.1: Showing Boss Information 📝

Found in: LevelSelectState.py

Define a dictionary called `boss_abilities` that maps each Boss Blind's name to its special effect. Each key should be the name of a boss (e.g., "The Mark", "The Needle", etc.), and each value should describe. This dictionary will later be used to look up and apply special effects based on which boss is active. You can use the tables at the beginning of this document to find this information.

Task 9.2: Boss Blinds Hand Debuff 🤌

Found in: LevelSelectState.py

Adjust player limits and reset values based on the current Boss Blind. Implement conditional logic that modifies the player's hand and discard limits depending on which boss is active. Finally, make sure to reset the player's round score to 0 at the end of this setup. Avoid unnecessary repetition, use clear condition structure to make the logic readable.

Bonus 🌟

If you're aiming for bonus points, feel free to enhance your game by adding new features and making it exceptional. Bonus points range from 1 to 10, based on the TA's discretion.

Some ideas:

- Add new jokers
- Add tarots
- New Game Mode
- Save system
- Finish debug menu

💡 **Important** ***Submit your project by November 21st at 11:59 PM to earn 10 bonus points*** *for your effort and punctuality! Taking advantage of this early submission bonus is a great way to boost your grade and show your dedication.*

Final Submission 🎯

To submit your project each team member must submit an evaluation form independently under PA3 in Moodle. One of the fields in the evaluation form will ask for a GitHub tag for your code. To generate a tag follow the following steps:

- Go to your repo and click on the Releases link usually found under the repository name.
- Click on the Draft a new release button.
- Under the Tag version field, enter the name of the tag you want to create (e.g v1.0.0).
- Fill in the release title and description (optional).
- Click on Publish release to create the tag and release.
- Enter the name of the tag in the evaluation form.